Overview

In this assignment, you will implement a simple web proxy that passes requests and data between a web client and a web server. This will give you a chance to get to know one of the most popular application protocols on the Internet- the Hypertext Transfer Protocol (HTTP)v. 1.0- and give you an introduction to the Berkeley sockets API. When you're done with the assignment, you should be able to configure your web browser to use your personal proxy server as a web proxy.

The Basics

Your task is to build a basic web proxy capable of accepting HTTP requests, making requests from remote servers, caching results, and returning data to a client.
This assignment can be completed in either C or C++. It should compile and run (using g++) without errors or warnings, producing a binary called proxy that takes as its first argument a port to listen from. Don't use a hard-coded port number (e.g., port 80).
You shouldn't assume that your server will be running on a particular IP address, or that clients will be coming from a pre-determined IP.

Listening

When your proxy starts, the first thing that it will need to do is establish a socket connection that it can use to listen for incoming connections. Your proxy should listen on the port specified from the command line, and wait for incoming client connections.
Once a client has connected, the proxy should read data from the client and then check for a properly-formatted HTTP request. An invalid request from the client should be answered with an appropriate error code.

Parsing the URL

Once the proxy sees a valid HTTP request, it will need to parse the requested URL. The proxy needs at most three pieces of information: the requested host and port, and the requested path.

Checking the Cache

After determining which web object is being requested (as named by the object's full URL), you should check to see if this object is already cached on the server. If so, you should return the content from the cache. **For simplicity, you do not need to implement proper HTTP expiry: You can simply clear your cache on bootup but cache objects indefinitely while the server is alive. You similarly do not need to support conditional GETS (e.g., "If-Modified-Since") to the remote origin server.** If desired, however, you can support real cache expiry.

Getting Data from the Remote Server

Once the proxy has parsed the URL, it can make a connection to the requested host (using the appropriate remote port, or the default of 80 if none is specified) and send a HTTP request for the appropriate file. The proxy then sends the HTTP request that it received from the client to the remote server.

Writing the Cache

After downloading a web object successfully, you should cache the object to disk so that subsequent fetches can use the local copy as opposed to fetching it again remotely. You should not cache the item if it is marked as "no-cache" or "private"; see the RFC. For this assignment, you only need to cache objects for requests that return type 200 (OK); you do not need to worry about other cacheable status codes such as 410 (GONE).

Returning Data to the Client

The proxy should send the response message to the client via the appropriate socket. Once the transaction is complete, the proxy should close the connection.

Testing Your Proxy

Run your client with the following command:
./proxy <port>, where port is the port number that the proxy should listen on. As a basic test of functionality, try requesting a page using telnet:

```
telnet localhost
<port> Trying 127.0.0.1...  Connected to localhost.localdomain
(127.0.0.1).  Escape character is '^]'.  GET http://www.google.com/
HTTP/1.0
```

If your proxy is working correctly, the headers and HTML of the Google homepage should be displayed on your terminal screen. Notice here that we request the full URL (http://www.google.com/) instead of just the absolute path (/). Your proxy should support both of these formats.
For a slightly more complex test, you can configure your web browser to use your proxy server as its web proxy.